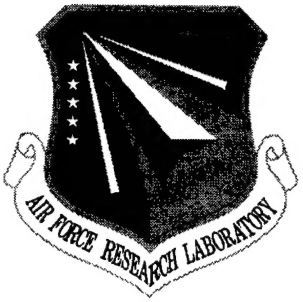
AFRL-IF-RS-TR-1999-150
Final Technical Report
July 1999



THE COMPLEX EVENT DETECTION AND MONITORING SYSTEM

Microelectronics and Compuer Technology Corporation

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F148

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-150 has been reviewed and is approved for publication.

APPROVED:

WAYNE A. BOSCO
Project Engineer

FOR THE DIRECTOR:

NORTHRUP FOWLER, III, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

THE COMPLEX EVENT DETECTION AND MONITORING SYSTEM

Donald Baker
Anthony R. Cassandra
Mosfeq Rashid

Contractor: Microelectronics and Computer Technology Corporation
Contract Number: F30602-97-C-0214
Effective Date of Contract: 10 June 1997
Contract Expiration Date: 31 March 1999
Short Title of Work: Surveillance of Complex Events Using Active Agents
Period of Work Covered: Jun 97 - Mar 99

Principal Investigator: Donald Baker
Phone: (512) 338-3362
AFRL Project Engineer: Wayne Bosco
Phone: (315) 330-3578

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Wayne Bosco, AFRL/IFTB, 525 Brooks Rd, Rome, NY.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jul 99	Final Jun 97 - Mar 99

4. TITLE AND SUBTITLE

THE COMPLEX EVENT DETECTION AND MONITORING SYSTEM

5. FUNDING NUMBERS

C - F30602-97-C-0214
PE - 62301E
PR - TCOM
TA - T2
WU - 06

6. AUTHOR(S)

Donald Baker, Anthony R. Cassandra and Mosfeq Rashid

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, TX 78759-5398

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

AFRL/IFTB
525 Brooks Road
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1999-150

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Wayne Bosco, IFTB, 315-330-3578

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Composite Event Detection and Monitoring System (CEDMOS) addresses a need for a general, domain independent event processing technology. CEDMOS recognizes patterns of events called complex events according to user-authored event specifications. CEDMOS is a general event processing technology that includes: 1) a core infrastructure for event detection which implements a general, efficient event processing mode; 2) a graphical programming environment for the creation and manipulation of composite event specifications; 3) a detector generator, which takes composite event specifications and generates Java(TM) code to recognize the specified composite events; 4) agent shells for rapid development of customized agents for event gathering, composite event detection, and dissemination of composite events. CEDMOS was developed for DARPA by the Microelectronics and Computer Technology Corporation (MCC). This paper gives the theoretical basis for the CEDMOS event processing model. The model is a restriction of a more general event processing model that into consideration a number of practical issues. In addition, issues that arose in the deployment of CEDMOS to some particular domains are discussed. Unlike many other event processing technologies, CEDMOS is not tied to databases or other technologies and can be applied to many different domains.

14. SUBJECT TERMS

Event Processing, Composite Events, Event-Condition-Action (ECA) Rules, Awareness

15. NUMBER OF PAGES

72

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Abstract

The *Composite Event Detection and Monitoring System* (CEDMOS) addresses a need for a general, domain independent event processing technology. CEDMOS recognizes patterns of events called *complex events* according to user–authored event specifications. CEDMOS is a general event processing technology that includes:

- a core infrastructure for event detection which implements a general, efficient event processing model;
- a graphical programming environment for the creation and manipulation of composite event specifications;
- a detector generator, which takes composite event specifications and generates Java[1] code to recognize the specified composite events; and
- agent shells for rapid development of customized agents for event gathering, composite event detection, and dissemination of composite events.

CEDMOSwas developed for DARPA by the Microelectronics and Computer Technology Corporation (MCC).

This paper gives the theoretical basis for the CEDMOS event processing model. The model is a restriction of a more general event processing model that takes into consideration a number of practical issues. In addition, issues that arose in the deployment of CEDMOS to some particular domains are discussed. Unlike many other event processing technologies, CEDMOS is not tied to databases or other technologies and can be applied to many different domains.

[1] Java is a trademark of Sun Microsystems.

Contents

List of Figures

Preface

During the course of a military operation, there is a need to closely monitor the events that are transpiring in order to make timely and informed decisions. The events of interest are many and can originate from a wide variety of sources; e.g., intelligence reports, weather reports, satellite imagery, etc. The volume and variety of events results in an extremely challenging decision making problem, which entails more than simply monitoring for the occurrence of a particular event or a simple set of events. Complex event patterns, where each individual event can be temporally and spatially related to the others, can be the crucial items the decision maker needs. However, these patterns of events can often be masked due to the immense volume of information that flows to the decision maker.

As a simplified example (see Figures 1 through 3), consider the simplified problem of trying to detect whether or not a particular region is in danger of being successfully invaded by the opposition forces. Suppose that there is a single event source which gives report of individual opposition force location and movements. We assume that the military commanders will have knowledge of the types and quantity of the opposing forces that would be required to mount a successful attack as well as their individual movement capabilities. Any opposition force that is smaller than this or which is too far away would not pose a serious threat. We further assume that the mere presence of an invading force itself is not enough to pose a serious threat to the commanders; without the corresponding proper intelligence (e.g., reconnaissance flights, scouting missions), the opposing force would be nothing more than a nuisance. In addition, the opposition's intelligence would also have to be recent enough to have relevance. Thus, the interesting event for the commander consists of some recent reconnaissance and scouting in a particular region followed by the gathering of an opposition force of sufficient size and in close enough proximity to a region.

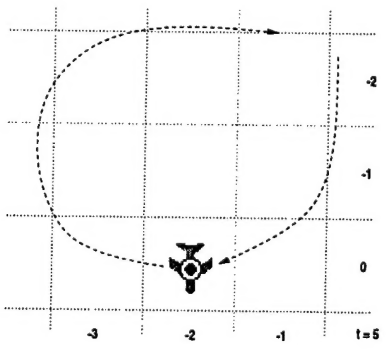


Figure 1: The initial event in the motivational problem: opposition reconnaissance aircraft is spotted over area $(0, -2)$ at $t = 5$.

Automated tools to assist in dealing with the abundance and variety of information would be extremely valuable, but this requires overcoming a number

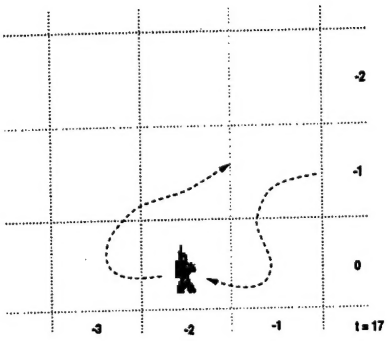


Figure 2: The second event in the motivational problem: opposition scouting troops spotted in area $(0, -2)$ at $t = 17$.

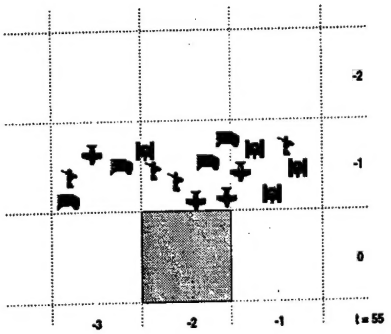


Figure 3: The final event in the motivational problem: an attack force of sufficient size becomes assembled within proximity to area $(0, -2)$ at $t = 55$.

of technical challenges. This paper describes methods for overcoming some of these obstacles and proposes techniques to apply to some of the other obstacles. The proposed system would monitor a large number of distributed, heterogeneous event sources and, given a set of the interesting event patterns, would identify the events matching these patterns. This work describes a general framework that encompasses problems which often appear under the heading of sensor fusion. However, the system described is not limited to any particular types of sensors or any particular domain. Nor are the characteristics of the problems addressed restricted to military operations; monitoring composite event patterns in the area of finance (e.g., stock price patterns) or security (e.g., intrusion detection) are two of the many alternative domains where this technology is applicable.

1 Summary

The *Composite Event Detection and Monitoring System* (CEDMOS) is a general, domain independent event processing technology. It addresses the need for recognizing patterns of events, called *complex events* according to user–authored event specifications. This paper first describes a theoretical basis for general event processing systems and then describes the CEDMOS event processing system relative to that theoretical basis. The theoretical model is embodied in a research prototype having an agent-based architecture. CEDMOS has been used in several practical settings. The application of CEDMOS to one such domain is described. CEDMOS is related to other event processing technologies The paper concludes with a discussion of future directions for CEDMOS development.

2 Introduction

Figure 4 shows the general world model on which we assume our composite event detection system must be built. On the left-hand side are the *event producers*, which are events generated outside of the detection system. In this example, we see a satellite, an Airborne Warning and Control System (AWACS), and general field intelligence reports being our sources of events. On the right-hand side we have various event consumers, or the people and systems, that would have interest in being notified of the individual events and various compositions of these events. The middle box is the *event processing system* which will contain the components necessary for defining, processing and delivering composite events. The Complex Event Detection and Monitoring System (CEDMOS) is a particular instance of an event processing system and the focus of this paper.

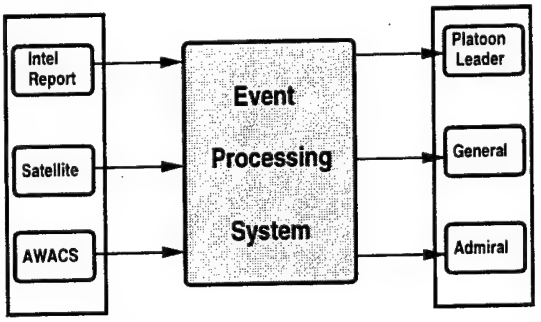


Figure 4: Simplified model of a typical event architecture.

Goals The goal of our work on CEDMOS is to develop an automated complex event detection system that is applicable to a variety of domains. Specifically, its goals are to:

1. design an expressive complex event specification model and associated language that can be used in a variety of practical settings (Ideally, the expressiveness and generality should not come at the expense of understandability and usability.);
2. build a working system to detect complex events specified in the complex event language (This includes graphical tools to facilitate the specification and visualization of complex events.);
3. apply the language, detector, and tools to several realistic settings and problems;
4. use the lessons learned to iteratively improve the language and related tools to begin to create a methodology of complex event specification; and

5. clearly delineate the inherent capabilities and limitations of event processing technologies in general and recommend the settings where they are best applied.

The last goal is important because it is easy to develop an unrealistic expectation about the capabilities of event processing technologies—that somehow, a complex event system will be able to do more than it was programmed to do.

Results The goals outlined above have all been accomplished to varying degrees, but there is still more work to be done to arrive at more satisfying conclusions. Specifically, the work accomplished and work to be done is as follows:

1. We have designed and complex event specification language that trades off expressiveness for simplicity. However, more work is needed to evaluate the effectiveness of this trade-off.

2. We have built a working system [5] based upon this language and constructed graphical tools for helping with the specification of composite events. The tools are prototypes and thus would need some refinement and added functionality for them to be generally useful.

3. We have applied this technology to two domains: battlefield awareness and a crisis response scenario. The battlefield awareness used a simplified simulator based upon the Modular Semi-automated Forcess (MODSAF) simulations. The crisis response application focused on the coordination of intelligence operations as they respond to world crises.[2]

4. With the two application areas used, we have obtained a great deal of insight and guidelines about the capabilities and limitations of event processing technologies. However, more extensive experience will be the only way to gain further insights and to evaluate its effectiveness.

Paper Outline Much of the previous work in complex event detection and modeling was in the active database research community. The database ancestry has biased the technologies so far produced. Some of the prior work relies on a number of tacit assumptions that do not extend to non-database settings. Because of our goal of generality, we will not assume that our technology will be used solely in a database setting. The remainder of this paper presents the technical rationale and details for our approach to composite event detection.

Section 3 discusses the terms and conceptual framework upon which CEDMOS is built: general event processing systems. This section also describes many of the key issues that arise in event processing and introduces the formal model of event processing. Section 4 then discusses the details of CEDMOS general event processing system terms. We begin this section defining our motivations in more detail and with our basic assumptions regarding the specific design decisions that

[2]Here CEDMOS was used as a component of a workflow system to provide general awareness of both event internal to the workflow process and external events.

were made in developing the CEDMOS model and software. Because we aim to have a direct connection between our general system and some real applications, we devote Section 5 to addressing the practical issues that arise with deploying CEDMOS and event processing systems in general. This section is based upon our work in developing real event processing applications. Finally, we outline the related work that has been done in event processing.

3 Events and Event Processing

This section defines the characteristics of what we call an *event processing system.* Many domain-specific systems often have one or more of these characteristics interleaved in their design. We have chosen to explicitly separate the event processing from a specific implementation to yield a model which can be used to provide critical functionality across a broad range of domains.

An event processor needs to address the following:

- **event definition** - what are the events dealt with by the system and how does the user map (syntactically and semantically) their particular domain events into those that can be used by the event processing system;
- **event gathering** - what are the specific mechanisms used to bring events into the event processing system;
- **composite events** - what are the mechanisms by which events can be combined to form events that are at a higher level of abstraction;
- **composite event specification** - how do users specify the composite events of interest, i.e., how do they use the mechanisms for defining combinations of events;
- **dissemination and presentation** - how are events exported out of the event processor and how are events presented to the user; and
- **event history** - what sort of mechanisms and control does the event processor offer to store and retrieve event histories of the running system.

Section 3.2 elaborates on each of these points, but before this we first present some of the concepts and terminology that will be required.

3.1 Event Model

Before discussing the details of an event processing system and the issues that crop up, we need to define some terms and concepts upon which we can base our discussion.

3.1.1 Event Definition

We begin with the following definitions:

Definition 3.1 Event — *an occurrence at a single point in time, either in the physical world or the virtual world inside a computer. An event is usually associated with some change of state.*

Definition 3.2 Primitive event — *a real-world event occurring outside the event processing system.*

These primitive events are the domain dependent entities that the event processing system is dealing with on the virtual level.

Definition 3.3 Base-level event — *a virtual event of the event processing system which corresponds to some external real-world event.*

An event processing system must begin by defining the events it will process and how they are are to be internally represented in the event processing system. Often, there is a direct correspondence between a primitive event and a base-level event (the external and internal representation of an event.) An event usually conveys more information than "the event occurred". We define this extra information as named fields, or *parameters*, of the event. These parameters can be any information about the context in which the event occurred. The time of the event's occurrence is considered to be a required parameter.

Definition 3.4 Event class — *a syntactic definition of a collection of information associated with the occurrence of a class of events. It is sometimes convenient to give the event type an explicit name. The structure and types of information for all events in a class are the same, but the actual instance values can differ.*

Definition 3.5 Event type — *synonomous with* event class.

Definition 3.6 Event channel — *a conduit for events conforming to a single event type from an event producer to an event consumer.*

Definition 3.7 Event stream — *a contiguous sequence of events of a specific type, such as might flow on an event channel.*

Event channels and event streams are convenient abstractions for describing what happens to events both inside and outside the event processing system.

3.1.2 Event Transformers

The most beneficial feature of an event processing system is its ability to recognize patterns in streams of events. Often, the individual primitive events do not directly convey meaningful information to the user, and many of these primitive events are completely irrelevant. The user is typically interested in higher-level events which are a composition of lower-level events.

Figure 5 shows a simple example of combining events. The situation is a long-range bombing mission where the pilot must decide whether to continue the mission when a particular way-point is reached. The assumption is that while the aircraft is in flight, the threats facing the bomber are to be neutralized, where the threats are assumed to consist of a radar installation and an airfield. While the aircraft is in flight, it will be notified of the outcomes of the other missions whose aims are to destroy these threats. These notifications, or reports, are the primitive events in this simple scenario. At the top level, the complex event

`abort-mission` will be generated if the threats are not neutralized and the way-point is reached. At the next level, the `threats-become-neutralized` event is generated if both the `radar-goes-down` and `airfield-becomes-unusable` events occur.

In Figure 5, the leaf nodes of the tree are the primitive events of the system and the internal nodes are used to capture some particular semantics of a combination of their input nodes. Roughly, the threats-neutralized node corresponds to a conjunction of the radar going down and the airfield becoming unusable and the `abort-mission` captures the sequential concept of the threats being neutralized before the way-point is reached.

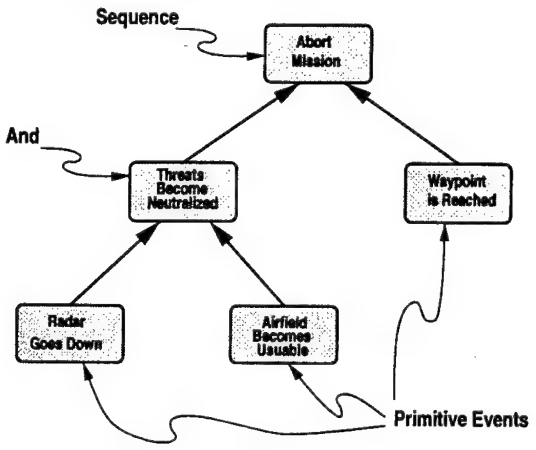


Figure 5: Simple example of a composite event.

As in Figure 5, we choose to view the construction of composite events as occurring in distinct computational pieces, which we refer to as *event transformers* and which appear as the internal nodes of the tree in the figure. Note that this is just a convenient perspective that does not limit the generality of an event processing system since even a monolithic system that does not explicitly decompose the event composition can be viewed as a system with a single, monolithic event transformer. However, the decomposition of the composite event computation into smaller pieces yields a system which is simpler, more flexible and more re-usable as we shall see when we discuss the CEDMOS event processing system in Section 4.

An event transformer takes one or more event streams as input and produces a single, higher-level event stream as output. Because the output is just an event stream, it can be further processed by another, higher-level event transformer. Arbitrarily complex event specifications can be constructed from combinations of event transformers.

3.1.3 Event Detectors

An *event detector* is simply a collection interconnected event transformers. Figure 6 shows this relationship and Section 3.3.3 discusses this concept further.

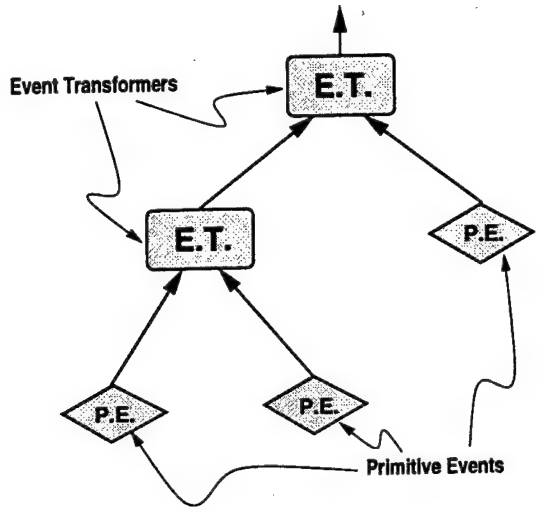


Figure 6: Simple example of an event detector.

3.1.4 Event Transformer State

We now return to the conceptual issues and complications that often arise when events are taken in combination. Recall that we defined an event to occur at an instant of time, having no duration. Careful analysis of Figure 5 when each arc "fires" for only an instant of time, shows that this combination of events *does not* truly capture the event that would be of interest to the pilot. When the bomber arrives at its way-point, it is not actually interested in whether it has received reports of the radar and airfield being rendered unusable; it is whether they are *currently* unusable that is the important question. The flight time of the bomber is extended enough that it is feasible for the radar installation or the airfield to be both disabled and repaired before the bomber reaches the way-point. Thus, we also need to consider events that are reports of the threats being repaired or replaced while the bomber is in flight.

Combining events in this way commonly gets misinterpreted since it becomes quite easy to assume that some form of implicit state exists in the boxes representing the complex event operators. Unlike a circuit, the "wires" connecting the components do not hold any state; they are simply a wire used to deliver an impulse that contains the event's information. For example, in Figure 5, one might be tempted to view the output of the operator labeled `threats-become-neutralized` as giving the current state of the threats. In fact, this operator only fires an event when both a report of the radar and

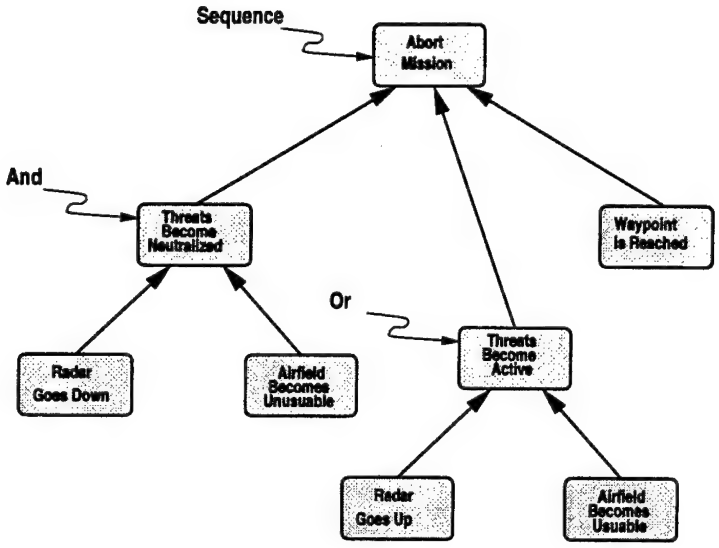


Figure 7: Alternative bombing run composite event.

airfield being unusable arrives. Since it is oblivious to reports of the facilities being repaired, this operator will fire if it receives the sequence of events `radar-goes-down`, `radar-is-repaired` and `airfield-becomes-usable`.

One could imagine fixing Figure 5 to incorporate these other events. Figure 7 shows such an arrangement, where the possibility of repair is taken into account. However, even here the event transformers requires some form of internal state to maintain the status of the threats. For the operators which fire based upon combinations of events (e.g., the "sequence" and "and" nodes), it must have some facility to remember which input events have fired thus far. However, even if we allowed the notion of the state of the threats in the internal node, what would this mean to its parent node? Would the parent node have a mechanism for querying the state of its children? When would this node fire off an event itself? The point here is that a state representation is somewhat at odds with an event–based approach.

Regardless of the answers, this shows that there is a great deal of subtlety can arise and that something more than a purely event-based model is needed. Because of this, the event processing system must admit to maintaining some form of state in its event transformers. Most existing composite event systems define *event operators* which impose certain specific semantics about how the inputs are combined. These define various forms of implicit internal state, that are often not discussed and which the user seldom has control over. We will see that CEDMOS gives the user the flexibility to control the state or to define event operators with implicit state.

3.2 Event Processing

The previous section has provided the terminology and some basic concepts we will use in this section as we discuss the main components and functionality that event processing systems need to address. The following sections elaborate on the main functions of event processing systems that were outlined at the beginning of this section.

3.2.1 Event Gathering

Having a general event processing system means that it must define events and operate at a particular level of abstraction. However, for the system to actually be deployable, it must provide mechanisms and tools for interfacing with the "real-world" primitive events and event sources that are available. This consists of both the physical and logical connectivity as well as the definitions of the translations from the primitive events to the base-level events. Because the physical connections are nearly always domain dependent, there is no escape from having to do this interfacing. A good analogy here are *device drivers* which are necessary to connect the hardware (external events) to a piece of software (the internal events).

3.2.2 Composite Event Processing

In general, an event transformer contains whatever computation is required to transform the input event streams into the composite event output stream. We choose to separate this computation into two main steps:

- **filtering** — the process of eliminating an event from an event stream, where the decision is based solely on the event's content;
- **summarization** — all other internal computations required to transform the input event streams into a composite event stream.

This computational division may or may not appear explicitly in the event processing system. Although not strictly required, event transformers will typically maintain some form of internal state as discussed in Section 3.1.4. Exactly how much state and the level of control given over this state are important characteristics of an event processing system and what can separate an effective system from a useless one.

An important aspect of the filtering computation is that although the criterion is imposed by the event transformer itself, because it only involves the event's parameters, it is possible to perform the filtering where the event is generated instead of where it is consumed. This can be an important optimization in systems where transporting the event from the source to the event transformer involves a non-trivial cost.

3.2.3 Composite Event Specification

With the ability to compose events into higher-level concepts comes the need for tools to define these abstractions. The most sophisticated event processor might be able to automatically construct these abstractions based upon some model of the underlying system and a model of the goals and desires of the user. However, simpler and currently more attainable systems will provide a means for the user to author composite event specifications.

Composite event specification involves the specification of a number of event transformers and the interconnections between them. Complete composite event specifications are usually constructed in a bottom-up fashion, starting with the base-level events. A base-level event stream is directed to the input of one or more event transformers through event channels. Then, some mechanism for defining the behavior of the event transformer is needed. The output of those event transformers may be further directed to other event transformers through additional event channels. Ultimately, a graph emerges that describes how each primitive event stream is combined into one or more composite event output streams. We will see later that it is useful to limit the allowable structures of these graphs.

One of the most important properties of an event processing system is the specific mechanism used to define the event transformer behavior. The internal computational model affects the computational properties of the system as a whole, while the method the system provides for defining a transformer has a profound impact on the usability of the system.

3.2.4 Composite Event Dissemination and Presentation

Similar to the need for the event processing system to provide support for connecting to the primitive event sources, is the need for it to provide mechanisms for exporting the composite events out of the event processing system. This dissemination is to a software component that has registered an interest in a particular composite event stream. The software component can then deliver the event as control signals to hardware components or as reports or alerts to humans.

When users are the ultimate consumers of an event stream, a number of human factors issues arise. For some composite events steams, each event is interesting, for other streams, only the latest event is of interest. Additionally, events may need to be queued and/or prioritized so that the user can attend to them as his time allows. Finally, the presentation of the event and its parameters can give rise to a number of complexities. For example, it is often desirable to correlate the event to relevant features in the event's domain.

As with the primitive event gathering, it is impossible to escape the domain dependent nature of the event delivery and presentation. However, an event processing system should strive to provide a general enough mechanism to allow it to be easily deployed in a variety of domains.

3.2.5 Event History

A useful component of an event processing system is the ability to record or log the events as they are received. This preserves a historical record of the system's operation and can aid users of the system is many ways. For composite events, the event history provides a means to "drill down," or "trace back" a high-level event to the low-level components from which it was derived. The event history can also be used to do post-mortem analysis of the system to evaluate and improve its operation or the manner in which it is used. The amount of information and the method of storing it depends upon the number of events and available resources. If all events cannot be logged, there needs to be a policy for deciding which event to save and which to ignore as well as a mechanism that allows users to control these policies.

3.3 Event Processing Model

Having provided the overall abstract view of event processing systems, this section takes a more formal approach. We will define the the event processing model with greater precision, since this will allow us in Section 4 to pinpoint exactly how CEDMOS is an instantiation of this, which allows a more rigorous analysis and comparison to other systems. However, this report does not frame existing systems in this model, and thus lacks the specific comparative analysis.

3.3.1 Time

An important issue that has to be addressed prior to developing the event processing model, is to discuss the nature of time, particularly in relationship to the assumptions about how events are allowed to occur.

Time Discretization The next issue is whether the event processing system views time as being continuous, or discrete. The general model does not restrict itself to discretized time, but assumes instantaneous results for computation. The current CEDMOS model assumes time is discretized and that any required computation for a given point in time can be completed before the next time step.

Event Generation vs. Event Reception Because message transportation time is not instantaneous, the time that an event occurs, can be quite different from the time that it is received for processing. Additionally, the distributed nature of the event sources means that issues concerning clock synchronization also arise. Distributed system research has explored this issue deeply and have a number of algorithms and techniques to address these problems. An actual real-time event processing system may need to adapt these techniques to provide the correct operational semantics of the system. In CEDMOS we deal only with the time associated with event generation and assume all the clocks are per-

fectly synchronized; an assumption which limits the applicability of the current CEDMOS prototype.

Event Simultaneity Another important time issue is whether or not two events can happen simultaneously. This applies to events coming from different event streams as well as to events within the same event stream. Our most general event model assumes that events can occur simultaneously, with a single event stream being able to produce more than one event for a given time. CEDMOS has rudimentary capabilities to support simultaneous events; it simply constructs an arbitrary ordering of the events, then processes them sequentially.

3.3.2 Event Transformer Model

Before developing the general model for an event processing system, we define the model for one of the most crucial elements required; the event transformer. In general, an event transformer can be any general function or more generally a Turing machine with inputs corresponding to the input events and the output corresponding to the composite event. However, we explicitly put a little more structure on the possible models. We define an event transformer as $\mathcal{M}^T = (\mathcal{E}^I, \mathcal{E}^O, \mathcal{S}, s_0, \Phi, \sigma)$ having the basic architecture shown in Figure 8. For simplicity of discussion, we assume all the sets we discuss with regard to this formal model are countable (i.e. isomorphic to the set of integers). We will see in a later discussion that most of these sets will be defined as a vector of typed values, which are also countable. While this theoretically eliminates an event processor from dealing with real-valued entities (which have the potential to be uncountably infinite), the finite precision inherent in computer hardware makes this assumption reasonable for practical systems. While the theoretical event processing model is described in terms of finite and countably infinite sets, in practice no infinite sets are ever needed or created.

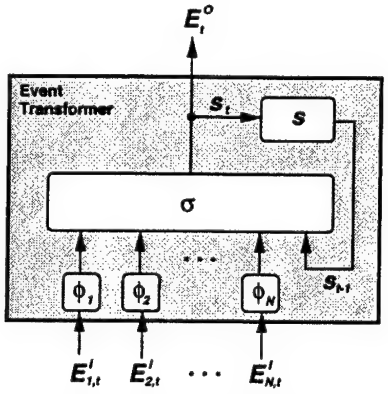


Figure 8: Basic event transformer model architecture.

The set $\mathcal{E}^I$ represents a vector of N input event classes, where each event class in the vector is not necessarily unique. Each element of this vector is an

input channel where a specific class of events can arrive. We use the notation $\mathcal{E}^I = (\mathcal{E}_1^I, \mathcal{E}_2^I, \ldots, \mathcal{E}_N^I)$ to show that the sets of input classes has an ordering. Non-uniqueness simply means that for some i and j, it is possible that $\mathcal{E}_i^I = \mathcal{E}_j^I$. A particular input to the event transformer at time t will be denoted

$$E_t^I = [E_{1,t}^I, E_{2,t}^I, \ldots, E_{N,t}^I]$$

where $E_{i,t}^I \in 2^{\mathcal{E}_i^I}$.[3] By allowing the inputs on each channel to be sets, we admit to the possibility of simultaneous events from the same input channel. An individual event at time t on input channel i will be denoted as $e_{i,t}^I \in E_{i,t}^I$.

The symbol $\mathcal{E}^O$ represents the class of possible events that can be output from this transformer. As with the inputs, we define an output at time t as $E_t^O \in 2^{\mathcal{E}^O}$ to show that multiple simultaneous output events can be produced by an event transformer. We let $e_t^O \in E_t^O$ denote an element of the output class at time t.

As previously discussed, the event transformer needs to maintain some form of internal state. The symbol $\mathcal{S}$ represents the set of possible internal states of the transformer. We also define the state, s_0, in which the transformer begins operation with $s_0 \in \mathcal{S}$.

Next, the model requires a set of filtering functions, $\Phi = \cup_{i=1}^N \{\phi_i\}$, one for each input channel, where $\phi_i : 2^{\mathcal{E}_i^I} \rightarrow 2^{\mathcal{E}_i^I}$ with the restriction

$$\phi_i\left(E_i^I\right) \subseteq E_i^I, \; \forall 1 \leq i \leq N \; .$$

This simply means that the filter function is a function of the inputs from one channel and that its sole function is to ignore some subset of the input events.

The final component in the event transformer model is the summarization function. In the general model, we impose no restrictions upon this function. At a given time t, it takes the input event sets and its previous internal state, s_{t-1} and produces a new state, s_t and a set of output events, E_t^O, which could be empty, i.e.,

$$\sigma : 2^{\mathcal{E}_1^I} \times 2^{\mathcal{E}_2^I} \times \ldots \times 2^{\mathcal{E}_N^I} \times \mathcal{S} \rightarrow \mathcal{S} \times 2^{\mathcal{E}^O} \; .$$

Alternatively,

$$\sigma\left(E_t^I, s_{t-1}\right) = \left(s_t, E_t^O\right) \; .$$

3.3.3 Event Detector Model

This section extends the event processing model by considering collections of event transformers. An *event detector*, $\mathcal{M}^D$, is a collection of one or more event transformers, and their interconnections. The interconnections define how the input and output event streams of the various event transformers relate to each other. Formally, $\mathcal{M}^D = (\mathcal{E}^I, \mathcal{E}^O, \mathcal{T}, \Omega)$, where $\mathcal{E}^I$ and $\mathcal{E}^O$ are the sets of input

[3] If A is a set, then 2^A represents the *power set* of A, or the set of all subsets of A.

and output classes of the detector, $\mathcal{T}$ is the set of event transformers in the model and Ω is a set of connections between the event transformers. (Note that when we defined an event transformer, $\mathcal{E}^I$ was a vector of possibly repeated input clasess, but in a detector, $\mathcal{E}^I$ is a non-ordered set. Additionally, for the event transformer, $\mathcal{E}^O$ was a single event class, in a detector it is a set of output event classes.) An example model is shown in Figure 9.

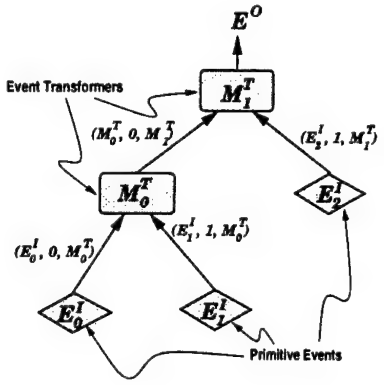


Figure 9: Example event detector.

The set of input event classes, $\mathcal{E}^I$, to a event detector must be all the input classes defined by event transformers in the set $\mathcal{T}$ which are not the output classes of other event transformers. The set of output event classes, $\mathcal{E}^O$, must be a subset of the output classes of the event transformers in $\mathcal{T}$.[4]

The set Ω has elements of the forms

$$\begin{aligned} &(\mathcal{M}_i^T, n, \mathcal{M}_j^T) \qquad \mathcal{M}_i^T, \mathcal{M}_j^T \in \mathcal{T} \\ &(\mathcal{E}_i^I, n, \mathcal{M}_j^T) \qquad \mathcal{E}_i^I \in \mathcal{E}^I, \mathcal{M}_j^T \in \mathcal{T} \ , \end{aligned}$$

where in both cases n is the positional number of one of the input event streams of event transformer $\mathcal{M}_j^T$. These interconnections essentially define edges of a directed graph where the nodes correspond to event transformers, inputs and outputs. Unlike a general graph though, we need to define an ordering on the incoming edges in order to properly connect the input and output event streams.

[4]You can also allow elements of the input set to be included in the output set, but we ignore this case for simplicity.

4 Methods, Assumptions, and Procedures

4.1 Assumptions and Motivations

Conceptually, composite event processing can be conducted by logging all events in a database or some other suitable representation and then expressing the composite events of interest in a query language or with some set of declarative rules. For domains where events do not occur too frequently and where the computational demands on evaluating the queries or rules are not too great, this might be a viable approach to composite event detection. However, often there are too many events, and the time-critical nature of the domain prohibits extensive computation. In these cases, an incremental event processing technology such as CEDMOS can be of great use.

Along with making design decisions that trade-off expressiveness for computational speed, the CEDMOS prototype has addressed a number of usability concerns, since the other main driving force is to develop a system that can be used by non-expert users.

Although we have designed and built a useful CEDMOS prototype, we have had to make some assumptions and simplifications about the nature of the domains in order to make the initial design task manageable. Below is a list of those assumptions; some are trivial implementation-specific assumptions, while others are made to avoid some deeper theoretical problems.

- We assume that each event class can be recognized by a globally unique event name.
- The information an event conveys can be represented by a flat list of attribute type-value pairs. There is no support for hierarchically organized information.
- All events have a time parameter which contains the time the event occurred.
- All computation inside an event transformer must be bounded and incremental in nature.
- The state inside a transformer must be defined explicitly. Although CEDMOS allows you to define more abstract *event operators* that have predefined state, in the basic model, the state must be defined.
- The detectors are passive (see Section 4.1.3).
- We assume that the composite event patterns are known (see Section 4.1.2).
- We have only addressed event simultaneity at a rudimentary level.
- We assume all clocks generating event time stamps are synchronized.
- We assume the time to transport and process events is negligible when compared to the inter-event arrival time.

4.1.1 Restricted Event Processing Model

The CEDMOS model represents a restriction of the general event processing model. Three primary goals drove our work on CEDMOS:

- *Usability*—the model should be easy to reason about and use;
- *Efficiency*—algorithms expressed in the model should have bounded computational complexity; and
- *Expressiveness*—the model should allow the capture of a wide range of event processing algorithms.

The goal of expressiveness is somewhat at odds with usability and efficiency. Expressiveness in the model allows algorithmic complexity that can be more difficult to use and more costly to compute. We have therefore tried to strike a balance between model expressiveness and the other goals.

The goals of usability, efficiency, and expressiveness led to a set of interrelated design choices in the creation of the CEDMOS event model. These choices include limiting the complexity of event transformers, partitioning the event transformer's state, a passive expression–based event transformer algorithm, and simplified handling of simultaneous events. Each of these design decisions is now discussed as they relate to the primary goals.

Restricting event transformer complexity promotes usability and efficiency at some expense of expressiveness. Simplified event transformers are not only themselves easier to reason about, but they promote reuse, thus improving user understanding of entire composite event specifications. Efficiency can be also guaranteed through computational restrictions on the event transformer algorithm. For example, a larger number event transformers may aid in the parallelization of the computational pipeline. While restricted event transformer complexity may be at odds with expressiveness, expressive composite event specifications can still be built from the building blocks of simple event transformers. Thus, our strategy is to force the complexity of composite event specifications to explicitly appear in the number of event transformers and their interconnections as opposed to being contained within the individual event transformers. Later, in Section 5, we will see that reusable, domain–specific event transformers with a fixed or parameterized algorithm, called *event operators*, are a natural place to hide much of the domain–specific complexities of composite event specifications.

The primary way we have reduced complexity of CEDMOS event transformers is by introducing the notion of partitioned event transformer state. The state of an event transformer is partitioned into disjoint, replicated finite units called *state categories*, with a single algorithm operating on each unit independently and in parallel. It is up to the creator of the event transformers to decide the space of possible state categories and what state will be kept for each state category. With categorized state, the event processing algorithm has four largely separate concerns: deciding whether an input event should be processed (*filtering*), selecting the state category based on an input event (*categorization*), updating the selected state category with information from an input event (*summarization*), and creating a composite event from the state category when one is to be emitted from the event transformer (*emission*). This separation of

concerns greatly simplifies reasoning about the event transformer and its computational cost without greatly reducing the overall expressivity of composite event specifications.

The next design decision is to embody the four steps of the event transformer algorithm into a passive, almost declarative form based on expression evaluation and computation based only on assignment and `if` statements. CEDMOS evaluates the expressions and assignments at the proper time.

- **Filtering.** Each input channel to the event transformer has an associated boolean expression over the parameters of the input event. If the expression evaluates to true, the input event is ignored.
- **Categorization.** The space of categories for an event transformer is determined by a finite set of named types, the number of which can be thought of as the dimension of the category space. Categorization is performed by a vector of *categorization functions*, one for each input channel to the event transformer. Conceptually, each such function takes an input event from the corresponding event class and maps it to a point in the category space. Computation proceeds only on that state category. In reality, the point in the category space is determined by a set of functions, one for each dimension in the space.
- **Summarization.** A state category is divided into typed fields not unlike a record in many programming languages. The summarization computation is a series of assignments to fields in the state category based on expressions over parameters of the input event and the prior values category state fields. In addition to these assignments, `if` statements are allowed. Iterating constructs, such as `while` loops, are prohibited.
- **Emission.** State parameter fields have optional properties either of being visible causing composite event firing. *Visible* fields become part of the output composite event, but *hidden* fields are private to the event transformer. The emission of a composite event is decided based on whether there is a net change to any *firing* field during the summarization computation. Generally, firing fields are visible because the reason for the composite event to be emitted would presumably be useful to a consumer. The emitted composite event will combine the named values defining the category, the prior values of the visible category state, and the updated values of the visible category state.

Specification of an event transformer's algorithm amounts deciding the input event types for each input channel, the category space, the state fields, and state field properties and the subsequent specification of filtering expressions for each input channel, categorization expressions for each dimension of the category space for each input channel, and summarization computations for each input channel. While there are a number of different kinds of things to be specified, their specification is straightforward. The appropriate expression or computation is invoked only as a result of the processing of a single input.

Consider an example where we have a number of temperature sensors in an office building, one per room, each sending their results every five minutes

according to a synchronized time signal. We would like to define an event transformer that computes a composite event that emits, every hour on the hour, the average temperature for several rooms of interest in the office building. We assume that all sensors emit events of the same event class with a parameter that uniquely identifies the sensor. The filtering function of the event transformer would have to distinguish whether the input event was from the sensor in a room of interest. If not, the event could simply be ignored. The category space of the event transformer might have two dimensions: one dimension for the room number and another, possibly, for the hour of the event. This category space will group together all inputs for the same room and for the same hour. Both dimensions of the category space can be directly computed from information we assume to be on the input event. Next, summarization involves computing a running average of the temperatures. The running average might be computed through saving a count of the number of events seen, the sum of the temperatures seen, and the temperature sum by the event count during each summarization. Finally, an event is emitted when the time of the input is the top of the hour.

The final design decision in the CEDMOS event model is the simplified handling of simultaneous events. Recall that the general event model allows processing of a number of simultaneous inputs on any combination of input event channels. In the CEDMOS event model, adopt a simplified, sequential event processing model. Multiple simultaneous input events are still allowed, but simultaneous inputs are serialized and processed as a batch. During the processing of the batch, no events are output. Instead, for state categories affected, the original state and current state are saved. At the end of processing, for those categories affected with differing state, a single event is output. This batching of inputs enables a form of "event coalescing" because while multiple simultaneous inputs may operate on a single state category, at most one output can be emitted per state category. The CEDMOS event model dictates that composite events are emitted as a result of any net change to the firing visible state after all simultaneous inputs have been processed. Event coalescing has the desirable effect of reducing the total number of events processed by the system through the suppression of extraneous events.

We close this section with a discussion of how the design decisions meet the CEDMOS primary goals of usability, efficiency, and expressiveness. The CEDMOS event model is usable because it breaks the event transformer specification process into a number of simple steps. The event model is easy to reason about because the basic algorithm is the same for all event transformers, with only the details of the filtering, categorization, update, and emission changing from one event transformer to another. Because we restrict all expressions to be simple functions and operating only on a finite state, we can place a constant time bound on the computational complexity of an event transformer processing a single event. With event coalescing and non–cyclic composite event specifications, the overall overall event processing algorithm is quite efficient. The expressivity of event model is harder to quantify. While we feel the CEDMOS event model is powerful, there are several restrictions that limit its expressivity.

The major expressivity limiting assumptions of the CEDMOS event model are finite category state, single categorization, and constant time summarization. The implications of each of these assumptions are now discussed.

- **Finite category state.** The assumption of finite category state prevents event transformers from saving additional information for each input event seen. The state computation must incrementally summarize in a fixed space information about the input events seen. Some summarization computations may not lend themselves to incrementality.
- **Single categorization for a given input event.** The CEDMOS model dictates that a single input event on a single input channel to an event transformer may contribute to only one category state. This limitation can be overcome by either adding another input channel in the event transformer from the same source with a different categorization function or splitting the event transformer into several simpler ones. We have yet to encounter a situation where neither of these strategies could solve the problem.
- **Constant time summarization computation.** In order to ensure an efficient event transformer algorithm the complexity of an event transformer's summarization computation is limited to constant time. This restriction prohibits an event transformer from embodying a complex computation. We have not found this restriction to be a serious one.

Overall, we feel that the CEDMOS event model meets the primary design goals of usability, efficiency, and expressiveness.

4.1.2 Monitoring vs. Discovery

Before delving into the issue of formally identifying the problem, we must first distinguish between discovering composite event patterns and monitoring events streams for composite event patterns. In this work, we view the composite event detection problem as one of monitoring event streams, while being *given* the specifications of interesting composite events. An alternative view would be to attempt extracting or *discover* the interesting composite events, given streams of events—effectively deriving the composite event specification. Obviously, the latter is a much harder problem. As with typical machine learning or data mining applications, this is a data intensive approach. Without hundreds or thousands of different event streams, extracting interesting event patterns would be near impossible. Although this could be feasible in the domain of finance where there are years of data are available, not all domains have enough existing data to make this a practical solution. Thus, this paper deals only with the approach of using specific domain knowledge to define the composite events. In addition, we focus more on the on-line scenario of incrementally detecting interesting events rather than the post-mortem analysis viewpoint.

4.1.3 Active vs. Passive

Another dimension along which event processors can differ is whether or not they actively seek out events and event sources, or whether they are passive and simply react to events being fed to them. The current CEDMOS prototype and model assume complete passivity and only perform computation in response to the reception of an event. However, the issue of actively seeking out events and event sources is mostly orthogonal to the internal event processing model. Thus, an active version of CEDMOS would not require significant changes to the underlying model.

4.2 CEDMOS Event Processing Model

The CEDMOS event processing model restricts the general event processing model in a number of ways. To enforce these restrictions, the actual CEDMOS computational model has to incorporate some additional complexity. The following sections formally define the CEDMOS computational model and put it in the context of the general event processing model of Section 3.3.

4.2.1 CEDMOS Event Transformer Model

Because the event transformer model of Section 3.3.2 is so general, practical instantiations need to impose restrictions and limitations on the expressiveness of the model. This quickly becomes the classic exercise of determining how to trade off generality with simplicity. This section defines exactly how CEDMOS has made these trade-offs.

The bulk of the added model complexity lies in the simplification of the inputs to the summarization function, σ. In the general model, because σ allows simultaneous inputs, the number of possible inputs to this function is a cross-product of a bunch of power sets of the inputs. Attempting to specify a function over such a large input space is much too cumbersome, and the CEDMOS system sacrifices some generality by restricting the domain over which the summarization has to be defined.

At the high level, CEDMOS breaks down the summarization function into two phases. The first phase consists of sequentially processing the elements in the inputs set to modify the internal state, while the second phase determines what the set of outputs should be, based upon the old and new state of the transformer. This reduces the complexity of the summarization function, since the summarization functions can now be specified over single events, rather than sets of events. What follows is the more formal definition of the CEDMOS transformer model.

Input Event Set The input event set for CEDMOS matches the input set for the general model. However, we will see that internally, when this set contains more than one element, CEDMOS will impose an ordering on the set and process each event sequentially.

Internal State CEDMOS imposes a special structure on the internal state and here we show the nature of that structure and relate it to the general model. Define a *finite* state set $\overline{S}$, then define $\mathcal{C}$ to be a possibly infinite set of *categories*. Assume that for each category $j \in \mathcal{C}$, there is a replica of this state set $\overline{S}$ denoted by $\overline{S}_j$.[5] We then define the full state for a CEDMOS event transformer as the cross-product, for each category, of these finite sets; i.e.,

$$S = \overline{S} \times \overline{S} \times \dots \ ,$$

where $\overline{s}_{j,t} \in \overline{S}$ will represent the component of the state for category j at time t. Thus, the state is partitioned into identically structured, finite pieces, where the number of partition elements is $|\mathcal{C}|$. Furthermore, CEDMOS defines a *categorization* function, κ_i, for each input channel $1 \leq i \leq N$ as

$$\kappa_i : \mathcal{E}_i^I \to \mathcal{C} \ .$$

We will see that each input can only affect the state within a single category at a time, and this mapping function helps define which category will be affected by a given input from channel i.

The initial state in CEDMOS is similarly defined in terms of categories. Letting $\overline{s}_0 \in \overline{S}$, we define CEDMOS's initial state as:

$$s_0 = (\overline{s}_0, \overline{s}_0, \dots) \ .$$

The same initial state must be used for all categories.

Filtering Function The filtering functions, ϕ_i, for CEDMOS are simplified to take a single input at a time, i.e.,

$$\phi_i(E_{i,t}^I) = \bigcup_{e \in E_{i,t}^I} \{\overline{\phi}_i(e)\} \ .$$

Here we have $\overline{\phi}_i : \mathcal{E}_i^I \to (\mathcal{E}_i^I \cup \{\emptyset\})$ with the further restriction that

$$\overline{\phi}_i(e) \in \{e, \emptyset\} \ ,$$

where $\overline{\phi}_i(e) = e$ for unfiltered events and $\overline{\phi}_i(e) = \emptyset$ when the event should be filter out. We will see later that CEDMOS requires the filtering functions, $\overline{\phi}$, to take on a simple form, that must have a computation time will be at most linear in the size[6] of the input.

[5]Note that even though the theoretical discussion of the CEDMOS event processing model relies on infinite state, in the actual system, the state is finite, but unbounded. Memory is only allocated for active state categories.

[6]The size of the input is taken to be the number of bits required to encode the input.

Summarization Function We now explain how the summarization function in CEDMOS is decomposed into a series of related functions. Since this requires processing the set of inputs sequentially, it will be convenient to define the size of the input set as:

$$M = \sum_{i=0}^{N} \left| E_{i,t}^I \right| \quad .$$

The first phase of the summarization function will require M stages of computation, one for each input element. Note that because we will only refer to the inputs at time t in this section, we have omitted the subscript from M, though it will differ over time.

We now break down the summarization function to incorporate this sequential processing[7] of the input events. Define the function $\rho(E_t^I, p)$ with $1 \leq p \leq M$, to be a function which assigns a single element from the input event set for each integer number p. Thus we have the constraints

$$\begin{aligned} \left| \rho(E_t^I, p) \right| &= 1 \quad \forall p, \text{ and} \\ \bigcup_{1 \leq p \leq M} \rho(E_t^I, p) &= E_t^I \quad . \end{aligned}$$

We then use an inductive definition for the series of intermediate states that occur while the serialized input events occur:

$$\begin{aligned} \widehat{s}_0 &= s_{t-1} \\ \widehat{s}_p &= \widehat{\sigma}\left(\rho\left(E_t^I, p\right), \widehat{s}_{p-1}\right) \quad . \end{aligned}$$

When the induction stops we have computed the value of the resulting state, $s_t = \widehat{s}_M$. The use of the function $\widehat{\sigma}$ instead of the general model's σ greatly simplifies the definition of the summarization function by limiting it to consider a single input at a time, i.e., $\widehat{\sigma} : \mathcal{E}_i^I \times \mathcal{S} \rightarrow \mathcal{S}$. However, since the state set, $\mathcal{S}$, can be infinite, we want to further simplify the summarization function to have its arguments be a single event *and* a single category state. If we define the category state subsets for the intermediate states such that

$$\widehat{s}_p = \left(\widehat{s}_{p,1}, \widehat{s}_{p,2}, \ldots, \widehat{s}_{p,|C|}\right) \quad ,$$

and let $e_p = \rho(E_t^I, p)$ where e_p is assumed to be from the input channel i ($e_p \in E_i^I$), then we can define each components' value with

$$\widehat{s}_{p,j} = \begin{cases} \overline{\sigma}_i\left(e_p, \widehat{s}_{p-1,j}\right) & \text{if } j = \kappa_i(e_p) \\ \widehat{s}_{p-1,j} & \text{otherwise} \quad , \end{cases}$$

The function $\overline{\sigma}_i$ is exactly what needs to be defined in the CEDMOS system. It is a series of functions, one for each input channel, that is defined over a

[7] The order of serialization is not under the user's control in the current prototype.

single input event and a single category state, i.e., $\overline{\sigma}_i : \mathcal{E}_i^I \times \overline{\mathcal{S}} \rightarrow \overline{\mathcal{S}}$. Practically speaking, it is much more convenient to be able to have a restricted domain for the summarization function, and as we will discuss in a later section, CEDMOS sacrifices expressiveness for computational speed in these $\overline{\sigma}_i$ functions to give it some of the performance guarantees it needs to be a responsive in a real-time environment.

To relate this back to the general model, we have

$$\sigma\left(E_t^I, s_{t-1}\right) = \left(s_t, E_t^O\right) \quad ,$$

where s_t is computed from the inputs and the previous state s_{t-1} according to the serialization of the inputs and the individual changes to the intermediate category states as described above. The only thing left is to define the outputs for the CEDMOS model.

Event Output Function Having presented the model for the incremental computation of the inputs, we now define the output set of a CEDMOS transformer to be computed by the function $E_t^O = \tau(s_{t-1}, s_t)$. Here s_t is computed using the mechanisms previously described. The essence of the τ function is to compare the old state and the new state and to generate an event if based on some difference between these. This change is detected on a category-by-category basis, but only for some subset of the category state. This is described in more detail below.

Let $\tau : \overline{\mathcal{S}} \times \overline{\mathcal{S}} \rightarrow 2^{\mathcal{E}^O}$ which defines the output of the event transformer. This function takes in two states, the state at time $t-1$ prior to the processing of events, and the state at time t, which is the resulting state after all the inputs have been sequentially processed. The output of τ is some subset of the set of possible output events, $\mathcal{E}^O$, which we define next.

CEDMOS breaks up the individual category states into two components: $\overline{\mathcal{S}}^V$ and $\overline{\mathcal{S}}^H$, the *visible* and *hidden* state components, and where $\overline{\mathcal{S}} = \overline{\mathcal{S}}^V \times \overline{\mathcal{S}}^H$. This decomposition of states will exist for each category and are also defined to be related to the output of the transformer by

$$\begin{aligned} \mathcal{E}_j^O &= \left\{\{j\} \times \overline{\mathcal{S}}_j^V \times \overline{\mathcal{S}}_j^V\right\} \text{ and} \\ \mathcal{E}^O &= \bigcup_{j \in \mathcal{C}} \mathcal{E}_j^O \ . \end{aligned} \tag{1}$$

Essentially, this just captures the fact that the output event of a transformer will consist of the old and new visible states of a particular category, j, as well as the category element itself, i.e., $e_t^O = (j, \overline{s}_{j,t-1}^V, \overline{s}_{j,t}^V)$.

We can then define the output of a CEDMOS transformer with

$$\tau\left(s_{t-1}, s_t\right) = \left\{\left(j, \overline{s}_{j,t-1}^V, \overline{s}_{j,t}^V\right) \left| \overline{s}_{j,t-1}^V \neq \overline{s}_{j,t}^V \ , \forall j \in \mathcal{C} \right.\right\} \ .$$

Note that if the number of categories is infinite, then so too is the output event set. However, at a given time t, the number of possible outputs is bounded by

$$|E_t^O| \leq \min\left(|E_t^I|, |\mathcal{C}|\right) \quad ,$$

since each category can produce at most one event and since no more than $|E_t^I|$ category states can change after processing the inputs.

Figure 10 gives a rough depiction of the CEDMOS event transformer model. It does not depict the serialization of the input events, but shows the fundamental relationship is has to the general model.

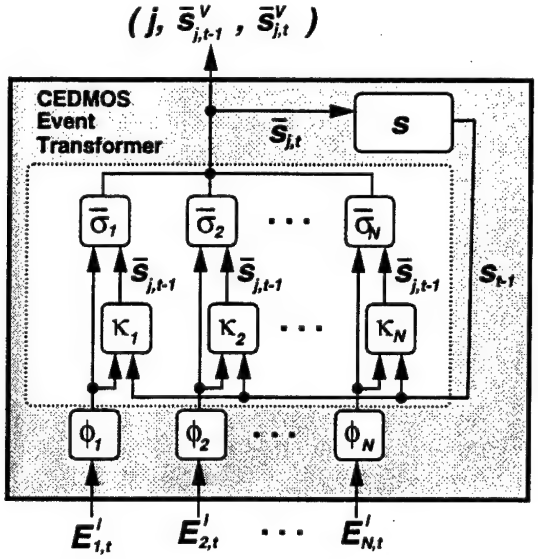


Figure 10: CEDMOS event transformer model architecture.

4.2.2 CEDMOS Event Detector Model

The event detector model for CEDMOS is identical to the general case presented in Section 3.3.3. However, CEDMOS imposes restrictions on the possible graph configurations allowed. Most importantly, only Directed Acyclic Graphs (DAGs) are allowed. Cycles would effectively add loops or recursion into the processing which adds a great deal of complexity to the internal processing model, as well as having the potential for non-trivial computation times. This naturally limits the computational power available, but greatly simplifies the composite event specifications and provides nice computational bounds for the entire detector.

Computation Time and Ordering In Section 3.3.3, the issue of computation time was ignored, as it was throughout Section 3.3. However, being a practical system, CEDMOS has had to deal with the issues of computation time, though it has only done so at a rudimentary level.

As primitive events occur, the event transformers which take those input streams can begin computing the values of their outputs (if any). If the outputs of these transformers feed into other transformers, then these transformers must

wait until the lower level transformer has finished computing. In this way, computation must precede in levels, with the the capability for parallel computation within a level. Figure 11 shows this pictorially.

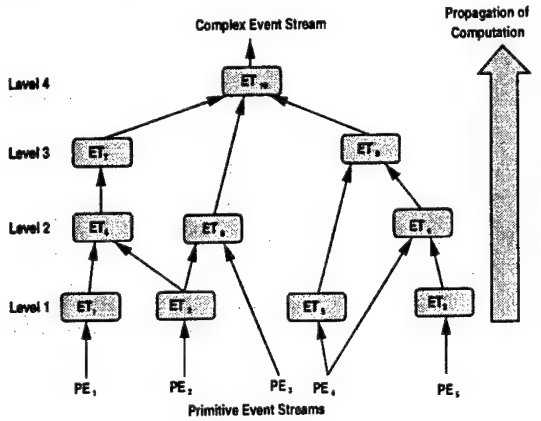


Figure 11: Structural layout of a sample composite event detector.

The fact that CEDMOS serializes simultaneous inputs means that the level above cannot commence until it has finished processing all of the events. Because the computation acts like a wave propagating through the system, it is possible for lower levels to begin processing the next inputs while the upper levels compute the previous computation; i.e., a pipelining of input events. While the CEDMOS prototype has rudimentary support to handle the propagation of computation, there is still more work to be done in this area.

4.3 CEDMOS Agents

Although the CEDMOS event processing model can be embedded within a software system, a significant amount of leverage and generality can be achieved by implementing various components as stand-alone pieces of software with the proper interfaces for communicating with the other components and external systems, i.e., using an agent-based architecture.

The CEDMOS architecture is capable of distributed event processing through a set of autonomous agents. These agents are responsible for introducing primitive events into the CEDMOS system, using them to create complex events, and acting as consumers of these events. They communicate through a mechanism called **JavaSpaces**[8], which is a modified version of *Linda Tuple Space* [3].

An *event source* agent converts primitive events happening outside CEDMOS into base-level events of specific event classes. Such agents can be connected to one or more external event sources and can produce one or more event streams. An *event consumer* agent represents one or more applications which will consume events. An agent with both the functionality of the event source and event

[8]**JavaSpaces** is a trademark of Sun Microsystems.

consumer agents can both inject primitive events into CEDMOS and consume events on behalf of an application. In this case, the application interact with CEDMOS to carry out event composition. Figure 12 illustrates how **JavaSpaces** and various CEDMOS agents stand in relation to each other.

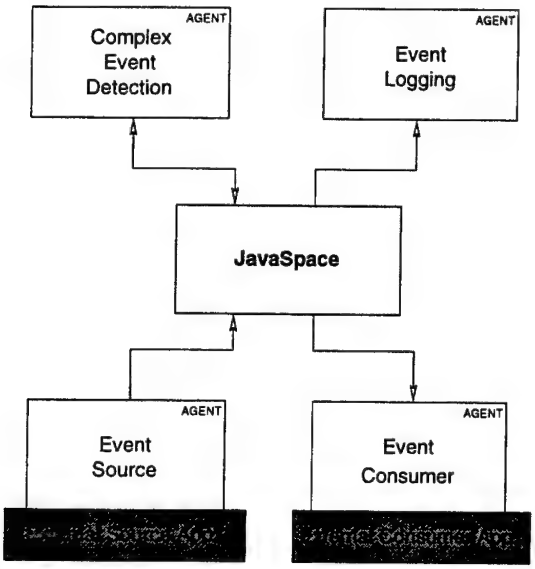


Figure 12: CEDMOS agent architecture.

The *complex event detection* agent consumes events and generates one or more different types of complex events. It is a wrapper around a collection of connected CEDMOS event transformers, and is otherwise known as a CEDMOS *event detector agent.* Such an agent takes multiple inputs and generates several different output events. A subset of the inputs are usually used by each event transformer, the output of which can be used by another event transformer within the agent or it can be directly tied to one of the outputs (see Section 3.1.3).

All the agents interact with each other through a **JavaSpace**. An event source agent can advertise, to the **JavaSpace**, the event classes associated with events it can produce, and the agent sends the generated events to it for the agents interested in consuming them. Any event consumer agent can read these advertisement, determine when it is necessary to subscribe to such events and consume them when they arrive at the **JavaSpace**.

There is also an *event logging* agent which logs all the events posted to the **JavaSpace** by the event producers. Its primary purpose is to keep a history of all the events generated by given instance of CEDMOS. In addition, it is designed to initialize an agent with previous events by replaying them through another temporary **JavaSpace** and then connecting it to the main one.

The advantage of such an architecture is the ability to dynamically introduce

or remove agents from the system and the flexibility to be run on different networked hosts. New event consumer agents can be added later to the system to consume existing events available in the running instance of CEDMOS. Similarly, agents representing newly available event source can connect to CEDMOS. It is also possible to create complex event detection agents to use existing and newly introduced primitive events and generate new complex events. Thus CEDMOS can evolve dynamically with the changing needs of multiple applications.

5 Results and Discussion

CEDMOS is a general, flexible infrastructure for event processing. General event processing, unfortunately, is too unwieldy to be used effectively "out of the box"—especially by end users. When customized for a particular application setting, however, the power of CEDMOS can be realized. This section primarily addresses the issues that arise when system developers customize CEDMOS for particular application settings, some of which may allow end users to interact with CEDMOS components. Customizing CEDMOS for a particular setting does require some work, much of which is conceptual in nature.

We make the primary assumption in this section that CEDMOS will be used in a setting in which it is appropriate: the near real–time processing of primitive events from a variety of concurrent, distributed event sources according to multiple composite event specifications. It is through those specifications that interesting patterns in the primitive events can be monitored and summarized in the form of composite events. Event processing can both raise the level of abstraction of events and reduce the volume of information that will flow to event consumers. Raising the level of abstraction simply means that the events' information can be closer to what the consumers are really interested in knowing. Reducing the number of events has the obvious benefit of easing users' information overload.

In Section 5.1, we first discuss the features of the application setting that are most important to applying CEDMOS to it. In Section 5.2, we next describe the general aspects of CEDMOS customization in high–level terms. To make this discussion concrete, we then describe a specific situation to which CEDMOS was successfully applied in Section 5.3. Finally, Section 5.4 delineates between appropriate and inappropriate usages of CEDMOS and enumerates some appropriate areas for CEDMOS.

5.1 Application Constraints

CEDMOS was intentionally designed to be applicable to a variety of practical settings. While we have tried to make the application of CEDMOS as easy as possible, there is only so much that can be put into a general infrastructure without making restrictive assumptions about how it is to be used. As such, CEDMOS poses few such constraints. Constraints do, however, arise from requirements from the specific application setting, including end user requirements, architectural considerations, and the application's domain model. These constraints are discussed in Sections 5.1.1, 5.1.2, and 5.1.3.

5.1.1 End User Requirements

System developers must pay particular attention in the application of CEDMOS to settings in which end users will directly or indirectly interact with it. There are two common situations where users might interact with CEDMOS. First, if end users are the ultimate consumers of the composite events detected by CEDMOS,

some mechanism must be provided for users to be alerted to the occurrence of those events and be shown the associated information. Event delivery may need to involve some form of persistent queuing since events are ephemeral by definition and users are not always available at event delivery time.

CEDMOS processes events according to one or more composite event specifications. It must be decided who will author those specifications and when they will be turned into detector agents. Will the authors be the system developers or end users of the application? If it is the latter, some consideration should be given to how can their experience be made more pleasant. Editing composite event specifications does require some amount of specialized knowledge, but information about the application setting can be heavily leveraged to make editing easier. Finally, if specifications will be authored while the system is running, some extra work will need to be done to bring a new specification "on line" with any others.

5.1.2 Architectural Considerations

The overall architecture of the target application will dictate whether and how CEDMOS's agent architecture will be employed. If the target system is distributed among various processes and machines, CEDMOS's agent architecture will likely be useful for gathering primitive events from the various sources and distributing the detected composite events to the various consumers. A non–agent–based system may benefit from directly using one or more embedded detectors, thus avoiding the added expense of inter–agent event communication.

5.1.3 Application's Domain Model

The semantic model of the application can dramatically constrain the usage of CEDMOS. We call the semantic model of the application area the *application domain* or just *domain*. The domain is useful in *giving meaning* to both primitive and composite events. Primitive events are real–world occurrences that relate to fundamental features of the domain. The domain largely dictates the parameters of the primitive event types. Additionally, the domain dictates what combination of events are meaningful. For example, it may be meaningful to combine several events into composite events because they each refer to semantically related aspects of the model. Semantic relationships may include, but are not limited to containment and temporal precedence.

The application domain can have a large impact on the definition of allowable event operators and how they can be combined. *Event operators* are reusable, domain–specific event transformers with a fixed or parameterized algorithm. It is often the case that adding application domain constraints can greatly simplify the editing composite event specifications. Meaningless combinations of event operators can be flagged as erroneous or simply removed from the realm of possibility. Event operators are discussed further in Section 5.2.1.

5.2 Domain–Specific Customization of CEDMOS

Now that some of the general application issues have been discussed, we now focus on the customization of CEDMOS that is required for specific application areas. Generally, CEDMOS must be customized exactly in those few places where it interacts with the outside world. These issues were a subset of those discussed in Section 3, but we bring them out again here for further discussion. These issues include using the application domain to restrict the CEDMOS event model; making a specification editor for the restricted model; using CEDMOS's agent architecture to gather primitive events and disseminate composite events; and a number of minor issues that must be addressed in a running system.

5.2.1 Restricting CEDMOS's Event Model

The CEDMOS event model is really a programming language for event specifications. Effective application of CEDMOS usually involves restricting the event processing model for the particular application domain. In some situations, this restriction takes the form of a fixed set of fully specified composite event specifications authored by the system designers. In other cases, a restricted model takes the form of a simplified event specification language that leverages properties of the application domain model. When this simplified language is to be used by end users, some care must be taken to make it easy to use. Still other situations may call for a middle ground where features of the domain are used to create a restricted language to be used only by system designers.

Composite event specification naturally breaks into two activities: the specification of event transformer algorithms and the interconnection of event transformers to make complete composite event specifications. When a range of composite event specifications is called for by an application setting, we advocate that the event transformers be specified by system designers in the form of reusable event operators. An event operator can adapt, where appropriate, to different numbers of inputs and different input types when instances of the operator are used in various settings. Additionally, operator's algorithm can be parameterized to allow for more flexibility with a minimal increase in specification complexity experienced by the event operator's user. Once a palette of event operators has been created, end–users can then interconnect instances of them in the creation of composite event specifications. In using this strategy, which we call the *domain–specific event operator strategy*, much of the domain knowledge is hidden from end–users within event operator algorithms. Even in cases where system designers will author composite event specifications, their specification process will benefit from the hiding of event transformer complexity in event operators.

Using the domain–specific event operator strategy, the CEDMOS event model can be readily restricted in three interrelated areas: restricting the allowable event types, creation of reusable event operators, and limiting event transformer interconnection. These three types of restrictions are now discussed.

Event types constrain events that flow on the event channels from primitive

event sources, between event transformers, and to composite event consumers. An application area will restrict event types in all of these cases. A fixed number of primitive event types are usually well defined by the real–world event sources in an application setting. At least some event transformers (or event operators) must take primitive event types as input. Event transformers emit events of types that are directly related to the inputs. While it is possible for an event operator to dynamically change its output event type depending on its number of inputs and their event types, the output event type will generally conform to a well defined range of possibilities. Still other event transformers may potentially exploit ranges of types in their inputs. Proceeding in a bottom–up fashion, all possible event transformer interconnection and detector output event types can thus be fully characterized.

Simultaneous with the restriction of event types is the creation of event operators that will have those types as inputs and outputs. While the creation of event operators is something of a creative art, some general principles apply. In order for events to be useful, they must be *meaningful.* Meaning is ensured by always making events relative to some feature of the application domain. Most event operators will be given by the relationships between features in the domain model for which there are events. For example, if the model has a feature C that contains two other features X and Y for which there are events. An operator that combines events from X and Y might be said to generate composite events on the container C. Still other operators are generic in nature, such as sequence, conjunction, and disjunction. While these operators are generic in nature, they cannot be made domain–independent because parameter handling requires some reliance on the application domain. It is for this reason that CEDMOS lacks domain independent operators. Other event processing languages side–step this issue by not requiring that events be self–contained.

The allowable event transformer interconnections in a composite event specification are largely determined by the event types produced and consumed by event transformers. Even with this restriction, however, not all connections are meaningful. The restricted language of event specifications for an application domain may therefore have additional restrictions to prohibit meaningless specifications.

5.2.2 Editing Composite Event Specifications

Once the restricted composite event specification language has been defined, it is usually straightforward to provide a mechanism for editing composite event specifications. CEDMOS provides a generic specification editor that can be customized for this purpose. Details of this customization are covered in the CEDMOS Users Guide [5].

5.2.3 Primitive Event Gathering

Primitive events must be gathered at their point of occurrence through the use of CEDMOS event source agents described in Section 4.3. Generally, a developer

will start with the event source agent shell and customize it for one or more primitive event types. For each type, a method will be created in the agent to be called when the event is created. The method's sole responsibility is to translate the given parameters to the types expected on the event and then emit the event. (In a non–agent–based system, a dedicated detector can be fed primitive events directly through a similar mechanism.) Employing the agent requires an event's point of occurrence to be instrumented to call the agent. If the event source code is available for modification and written in Java[9], this instrumentation is a straightforward process.

5.2.4 Composite Event Dissemination

Analogously to the issue of gathering primitive events is the issue of disseminating detected composite events. Composite event dissemination involves the customization of the CEDMOS event consumer shell. Such customization requires the overriding of an abstract method to consume events for one or more event types for which the agent has registered interest. Usually the customization involves changing event parameter types to suit the needs of the consumer. At run–time the method will be called asynchronously as events are generated.

If the ultimate consumer of a composite event is an actual user, more complexities arise in the area of composite event dissemination. End users are generally not assumed to be always logged on to the system, so system developers must save the events in a persistent queue for consumption by the end user. In some situations a tabular display of events and their parameters may be completely adequate. Many deeper issues should be considered, however, such as relating similar events to each–other visually, event priorities, and relating event information back to the domain. These important topics are outside the scope of this report.

5.2.5 Run–Time Issues

The composite events detected by CEDMOS depend on the initial state of a detector and the primitive events "seen" by it. If the detector misses primitive events, for example, it will produce incorrect results. Some care must be taken to ensure that the detector is kept in synchronization with the real world primitive event sources.

In the ideal case, the lifetime of the detector fully encompasses the lifetime of the primitive event sources. When this is the case, the detector is guaranteed to receive all primitive events and therefore stay synchronized. The only complication in this situation is if the detector may be brought up and down. Here, the detector's state must be persistently saved between runs. CEDMOS provides a mechanism to store its state categories in an XML[10] format and read them in again. In some cases, it may be possible to bring the system to a base state

[9]Java is a trademark of Sun Microsystems.

[10]XML is a trademark of the World Wide Web Consortium.

where any event history is irrelevant. In such a case, detector initialization is unnecessary.

A more complicated situation arises if the detector is to be initialized to reflect the state after primitive events have occurred. CEDMOS provides a mechanism to log and replay event streams transmitted via the JAVASPACE. If it is known that the events entered the JAVASPACE, but not the detector, the events from the space can be replayed to the detector in the order they actually occurred. During this initialization process, some consideration must be given to whether detected composite events are appropriate for release to the remainder of the system. Once a detector has been initialized in this manner, it can then function normally for subsequent input events.

In the worst case, primitive events have occurred that are unseen by any part of CEDMOS. Here, the only recourse is to initialize the detector from knowledge about the domain and the current state of the world. If all event operators reflect meaningful composite events, then their state can be characterized through queries on the actions that have occurred. Unfortunately, because this knowledge is often not kept, or too expensive to compute, this form of detector initialization is outside the realm of possibility. When this knowledge is available, it is possible to initialize a detector by initializing each event transformer's category state based on the relevant knowledge of the current state of the world. Because the details of this process depend heavily on the application domain and the specification and configuration of event transformers, little more can be said on this topic in general.

5.3 A Case Study: Awareness Provisioning in the CMI Workflow Management System

CEDMOS has been successfully customized for providing awareness information in the Collaboration Management Infrastructure (CMI) advanced workflow management system. CMI is a research prototype developed at MCC that supports work on advanced coordination modeling, awareness provisioning, and the brokering over reusable, externally provided services. CEDMOS was used as part of CMI's awareness provisioning mechanism. *Awareness* is defined as information that is highly relevant to a user's situation and needs. CMI's awareness provisioning mechanism, described fully in [1] and more generally in [9], is built using an event processing approach. All workflow events are processed according to a number of *awareness specifications* that combine a restricted set of composite event specification with delivery instructions that define a class of users for whom the detected composite events are to be delivered. Awareness provisioning in CMI is used to keep users abreast of process–related information that is relevant to them. Presumably, through making users aware of relevant information in a timely fashion, they can make better decisions. We describe awareness provisioning in CMI as a case study in how CEDMOS can be effectively used in a particular application domain.

The remainder of this section mimics that of Sections 5.1 and 5.2. Here we discuss how the specifics of using CEDMOS event processing as part of CMI's

awareness provisioning mechanism.

5.3.1 Application Constraints

CMI's application constraints had a large impact on how CEDMOS was used. The application constraints derived from end user requirements, architectural considerations, and CMI's domain model.

End User Requirements CMI end users interact with CEDMOS both for consuming detected events and in authoring event specifications. One class of users, called *participants*, receive awareness information in the form of composite events, called *awareness events*. At event detection time, the event's delivery instructions are evaluated, resulting in a set of users who are to receive the information. For each user, the event is copied into a user–specific awareness event queue. While participants are logged on to the system, they are kept abreast of any change to their event queue. For any event they have received, they can view its parameters and invoke related actions.

Another class of users, *process designers*, author awareness specifications that describe what information is to be provided to what class of participants under what circumstances. Awareness specifications are a restricted form of composite event specifications with attached delivery instructions. Both the composite event specifications and delivery instructions draw from the underlying application model which is CMI's process model, also called the *Collaboration Management Model (CMM)*.

Architectural Considerations CMI's architecture loosely follows a client–server model with a set of interacting agents playing the part of the server. The heart of the server is the process engine which coordinates the activities of users based on a process specification. Other agents manage resource usage and service brokering. Many of these agents are primitive event sources instrumented with CEDMOS primitive event source agents. Primitive events flow to CEDMOS detector agent(s) that embody one or more awareness specifications. Detected awareness events must ultimately flow to the CMI client for consumption by users. An intermediate server, called the *awareness delivery server*, consumes all awareness events and maintains queues for each user. From CMI's perspective, all awareness processing is handled by a software component called the *awareness engine* which is comprised of a customized version of CEDMOS and the awareness delivery server.

Application's Domain Model CMI's application domain model is the Collaboration Management Model (CMM). CMM is an advanced process model that coordinates the work of participants according to a specified business process. The model contains a variety of event sources, including process and activity state change events, resource change events, and dependency enactment events. All events are in relation to an enacted *process instance* described

by a *process schema.* The process schema gives structure and meaning to the primitive events and composite events specifications derived from them.

5.3.2 Domain–Specific Customization of CEDMOS

CEDMOS did require some domain specific customization for use in CMI's awareness provisioning mechanism. These customizations are now described.

Restricting CEDMOS's Event Model CMI's awareness provisioning mechanism restricted the CEDMOS event processing model in a variety of ways. *Awareness specifications* in CMI are CEDMOS composite event specifications conforming to a restricted event model drawing fixed palette of operators combining events from a small number of primitive event sources. Each primitive event source must first be first processed by an event operator that filters events according to a particular feature of a previously defined process schema. The resulting composite event is then specific to that process schema. Events relative to a process schema can be combined together with user parameterizable event operators including sequence, conjunction, and disjunction. All such operators categorize over process instance so that information is not mixed across process instances. All events to be output from a detector must first pass through an *output event operator* that adds the awareness delivery instructions as extra parameters. With the exception of primitive events entering the detector and composite events to be output from the detector, all events have the same *canonical event type.* A uniform type reduces the number of constraints on operator interconnections in composite event specifications.

Editing Composite Event Specifications CMI's awareness specification editor is a customization of the CEDMOS composite event specification editor that embodies the restricted event model just described. In addition to defining a palette of event operators and small dialog–based GUIs for user editing of operator parameters, the editor manages the filing of awareness specifications in XML format and the generation of code for the corresponding CEDMOS detector agents.

Primitive Event Gathering CMI employs several customized event source agents for the gathering of primitive events from the various CMI components. All such agents are a straightforward use of the CEDMOS event source agent shell.

Composite Event Dissemination All detected awareness events conform to a single event type. The CMI awareness delivery server registers an interest in events of that type and consumes them on behalf of all users. Upon receipt of the event, the attached delivery instructions are evaluated resulting in a set of users. Information from the event is placed in a queue for each user. Through an interaction with the awareness delivery server, the CMI client for participants

tool can retrieve queued awareness events and be notified when new events arrive. The awareness delivery server is a customization of the CEDMOS event consumer agent shell. The awareness event queue management capabilities of the awareness delivery server and the awareness information viewer in the CMI client for participants are functions not provided by CEDMOS.

Run–Time Issues In general, workflow management systems are long lived systems that maintain a persistent state. The persistent state facilitates system reconfiguration and crash recovery. Because awareness specifications are authored at process build time, CMI adopts a strategy of keeping the CMI awareness engine running at all times the remainder of the system is running. Using this strategy guarantees that no events are missed by a composite event detector.

5.3.3 Evaluation

CEDMOS proved useful as the main component in CMI's awareness provisioning mechanism. This application is a non–trivial application of CEDMOS that makes full use of its capabilities. CEDMOS did require some customization for this application setting. The awareness editor required a total of about 9000 lines of code, most of which is for the twelve event operators and the associated operator parameterization dialog boxes. Another 2500 lines is required for CEDMOS detector agent run–time support in the CMI architecture, including primitive event gathering. The awareness event delivery server required 2000 additional lines of code. All of these numbers refer to Java code having a coding style with about 13% blank lines and 34% comments. Even though CMI's awareness provisioning mechanism involved extensive customization of CEDMOS, considerable effort was saved over not using CEDMOS at all.

5.4 Appropriate CEDMOS Application Areas

CEDMOS is appropriate for the near real–time processing of primitive events from a variety of diverse, concurrent, distributed event sources according to multiple composite event specifications. While events may arrive at any time from a distributed set of event sources, their arrival rate must not swamp the event processing algorithm. Section 3.3.1 discussed event processing time complexity in CEDMOS. Finally, because CEDMOS is coded Java, it is best suited in situations where primitive event sources and composite event consumers are also written in Java.

We now briefly discuss two areas where CEDMOS could be applied: network intrusion detection and distributed debugging.

5.4.1 Network Intrusion Detection

Network intrusion detection is the discovery of suspicious patterns of behavior based on information gathered at various points in a computer network.

Because of the myriad strategies for network break–ins, any intrusion detection algorithm must monitor for a large number of suspicious behavior patterns simultaneously. Additionally, since a break–in may involve multiple hosts, a distributed algorithm is essential. Much of the activity on a network can be viewed as discrete events where the associated information can be sent as parameters. CEDMOS's event processing capabilities are well suited to this application area.

5.4.2 Distributed Debugging

Debugging of interactive distributed computations is difficult for users because of the vast amount of information produced. Program execution can be characterized as a sequence of events with each event representing a statement or watch point. A person debugging a system has a mental model that characterizes the expected behavior of the system. Perhaps this expected behavior includes contracts, pre-conditions, post-conditions, and cause and effect relationships. Over time the user may characterize the system's correct behavior in terms of composite event specifications that trigger when these conditions are violated. Again, CEDMOS is well suited for this application.

6 Related Work

The active database community has long been investigating the application of the Event-Condition-Action (ECA) paradigm in the context of using triggers, generally associated with update, insert or delete operations. Event generation, condition evaluation and actions are all performed within the database. *HiPAC* [7] was of the first active database projects to define an event algebra To name a few more recent representatives, *Alert* [13] brought the event-based approach to passive databases, while a more sophisticated approach has been used in *REACH* [2] and *Ode* [8] for object-oriented active databases. CEDMOS complex event detection expands on the expressive event specification of the *SNOOP* [6] language developed as a part of the Sentinel project. This particular work explores the issue of event consumption policy. All of them investigate defining operators and specification for complex events. A good overview appears in [15].

A good example of a system level event detection using event-condition-action paradigm is the *Yet another Event-Action Specification Tool (YEAST)* [12] system from AT&T. It has a facility to specify user-defined events through an announcement mechanism, in addition to a set of predefined system events. Events are associated with objects, their attributes and relational tests on the attributes and time descriptors such as *in, by, at,* etc. Unlike the active database community, there are only a few simple operators like *or* and *and* to create complex events. The action part is specified in terms of *ksh* script, and elaborate operations can be carried out using this facility. A large part is implemented using polling, but some file system level modification allow direct access to YEAST. In spite of its sophistication, it is a monolithic server both handling requests from clients and checking the queues of polled event descriptors. Such an architecture, although suitable for one system, is unsuitable in large scale distributed event detection. Nevertheless, having access to system level events is a powerful concept, especially for distributed complex event detection.

Rapide [11] is an effort to bring the widely used event-based mechanism in simulation into the specification of language. It can be used as a concurrent event-driven simulation language to create and executable architectural definition of a system. It was inspired by the event-based architectural specification found in *Very High Speed Integrated Circuit Hardware Description Language (VHDL)* [10] used in hardware design. The event specification in Rapide, however, is more sophisticated and uses a partially ordered set of events instead of a linear trace. Once again, like other systems mentioned so far, it lacks some event composition operators and fails to address difficulties associated with event consumption and distribution in a large scale environment.

The notion of event notification is present in the *Remote Network Monitoring Management Information Base (RMON)* [14] extension of the popular Simple Network Management Protocol (SNMP) [4] approach using Management Information Base (MIB). Alarm and Filter objects on Channels defined in this MIB use the Event object to notify one or more monitoring stations on the network. These events are sent by the probes running at the remote devices. The pres-

ence of such a rudimentary event-driven approach contrasts sharply with the request for information by the monitoring stations and the response by the remote location. The locus of control here is at the remote source. The associated logging facility provides a powerful diagnostic capability. But it only addresses a narrow domain of network management using restrictive MIB mechanism, and the difficulties of event composition and large scale event consumption remain unaddressed.

7 Conclusions

This report has provided the basic issues associated with general event processing and has defined the CEDMOS model. Additionally, we have given our design rationale and noted the places where more theoretical work needs to be done. In our prototype we have addressed the practical issues with using event processing technologies, and have evaluated the CEDMOS model and software in the context of two realistic domains.

While there is still a few outstanding issues and model refinements that need to be done, the basic CEDMOS model and proof-of-concept demonstrations have shown that general event processing can be a useful tool for a range of domains. Future work on CEDMOS will likely be in the following major areas:

1. Providing a general monitoring and event trace-back capability that could be customized for various domains.

2. Gaining more experience with practical application of CEDMOS to novel domains.

3. Using the experience from such practical application to refine and extend the CEDMOS event processing model.

4. Extending the CEDMOS event processing model to allow for prediction of events.

5. Improving the scalability and performance of the CEDMOS agent architecture.

8 References

[1] Donald Baker, Anthony R. Cassandra, Hans Schuster, Dimitrios Georgakopoulos, and Andrzej Cichocki. The CMI approach for providing awareness in a process–oriented environment. Technical Report MCC–CMI–119–98, Microelectronics and Computer Technology Corporation (MCC), 3500 West Balcones Center Dr., Austin, Texas 78759, January 1999.

[2] A. P. Buchmann, H. Branding, T. Kurdass, and J. Zimmerman. REaltime, ACtive, and Heterogeneous mediator system. *IEEE Data Engineering Bulletin, Special Issue on Active Databases*, 15(4):44–47, December 1992.

[3] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[4] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Snmpv2 (several rfcs). RFC 1901–1908.

[5] Anthony R. Cassandra, Donald Baker, and Mosfeq Rashid. CEDMOS user's guide. Technical Report MCC–CEDMOS–011–99, Microelectronics and Computer Technology Corporation (MCC), 3500 West Balcones Center Dr., Austin, Texas 78759, February 1999.

[6] S. Chakravarthy, V. Krshnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the 20th Conference on Very Large Scale Data Bases*, pages 606–617, Santiago, Chile, 1994.

[7] U. Dayal, B. Buchmann, and S. Chakravarthy. *In HiPAC Project.* Morgan Kaufmann, New York, 1996.

[8] M. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an object-oriented data. In *Proceedings, International Conference on Management of Data*, pages 81–90, San Diego, California, June 1992.

[9] Dimitrios Georgakopoulos, Hans Schuster, Andrzej Cichocki, and Donald Baker. Collaboration management infrastructure in crisis response situations. Technical Report MCC–CMI–009–99, Microelectronics and Computer Technology Corporation (MCC), 3500 West Balcones Center Dr., Austin, Texas 78759, March 1999.

[10] IEEE Inc., Los Alamitos, California. *IEEE Standard VHDL Language Reference Manual, IEEE Standard 1076–1987*, 1987.

[11] D. C. Luckham and J. Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9):717–734, September 1995.

[12] D. Rosenblum and B. Kirshnamuthy. Yeast: A general purpose event-action system. *IEEE Transactions on Software Engineering*, 21(10):845–857, October 1995.

[13] U. Schreier, H. Pirahesh, R. Agrawal, and C. Mohan. An architecture for transforming as passive dbms into an active dbms. In *Proceedings of the 17th Conference on Very Large Scale Data Bases*, pages 469–478, Barcelona, Spain, September 1991.

[14] S. Waldbusser. Remote network monitoring management information base. RFC 1757.

[15] Detlef Zimmer and Rainer Unland. The formal foundation of the semantics of complex events in active database management systems. Technical Report 22/1997, C-LAB, Paderborn, Germany, 1997. Available at "`http://www.c-lab.de/~aatools/external/1997/cr97_22.ps.gz`".

9 Symbols, Abbreviations, and Acronyms

9.1 Acronyms

CMI Collaboration Management Infrastructure.

CMM Collaboration Management Model.

CEDMOS Composite Event Detection and Monitoring System.

DARPA Defense Advanced Research Projects Agency.

ECA Event–Condition–Action.

GUI Graphical User Interface.

MCC Microelectronics and Computer Technology Corporation.

XML Extensible Markup Language.

9.2 Symbols

$\mathcal{C}$ Set of CEDMOS state categories.

E, E^I, E^O Set of events, set of input events, set of output events.

$\mathcal{E}$, $\mathcal{E}^I$, $\mathcal{E}^O$ An event class, an input event class, an output event class.

$\mathcal{M}^D$ The model of an event detector.

$\mathcal{M}^T$ The model of an event transformer.

$\mathcal{S}$ The set of possible internal states of an event transformer.

$\overline{\mathcal{S}}$, $\overline{\mathcal{S}}^V$, $\overline{\mathcal{S}}^H$ CEDMOS category state set, visible category state set, hidden category state set.

e, e^I, e^O An event, input event, and output event.

s_t, s_0 Event transformer state at time t, the initial event transformer state.

$\overline{s}$, $\overline{s}^V$, $\overline{s}^H$ CEDMOS category state, visible category state, hidden category state.

$\widehat{s}$ Intermediate CEDMOS category state.

$\mathcal{T}$ Set of event transformers in an event detector.

Φ Set of event transformer input filtering functions.

Ω Event detector connection set.

κ Event transformer category mapping function.

ρ Event transformer input selection function.

σ Event transformer summarization function.

$\overline{\sigma}$, $\widehat{\sigma}$ CEDMOS summarization function, CEDMOS intermediate summarization function.

τ Event transformer trigger function.

ϕ, $\overline{\phi}$ A single event transformer filter function, a single CEDMOS event transformer filter function.

*U.S. GOVERNMENT PRINTING OFFICE: 1999-610-130-81163

DISTRIBUTION LIST

addresses	number of copies
AFRL/IFTB WAYNE A. BOSCO 525 BROOKS ROAD ROME, NY 13441-4505	6
MICROELECTRONICS & COMPUTER TECHNOLOGY CORP (MCC) 3500 WEST BALCONES CENTER DRIVE AUSTIN, TX 78759-6509	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
DR JAMES ALLEN COMPUTER SCIENCE DEPT/BLDG RM 732 UNIV OF ROCHESTER WILSON BLVD ROCHESTER NY 14627	1
DR YIGAL ARENS USC-ISI 4676 ADMIRALTY WAY MARINA DEL RAY CA 90292	1
DR MARIE A. BIENKOWSKI SRI INTERNATIONAL 333 RAVENSWOOD AVE/EK 337 MENLO PRK CA 94025	1

DR MARK S. BODDY
HONEYWELL SYSTEMS & RSCH CENTER
3660 TECHNOLOGY DRIVE
MINNEAPOLIS MN 55418

1

DR PIERO P. BONISSONE
GE CORPORATE RESEARCH & DEVELOPMENT
BLDG K1-RM 5C-32A
P. O. BOX 8
SCHENECTADY NY 12301

1

DR MARK BURSTEIN
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

1

DR THOMAS L. DEAN
BROWN UNIVERSITY
DEPT OF COMPUTER SCIENCE
P.O. BOX 1910
PROVIDENCE RI 02912

1

DR WESLEY CHU
COMPUTER SCIENCE DEPT
UNIV OF CALIFORNIA
LOS ANGELES CA 90024

1

DR PAUL R. COHEN
UNIV OF MASSACHUSETTS
COINS DEPT
LEDERLE GRC
AMHERST MA 01003

1

DR JON DOYLE
LABORATORY FOR COMPUTER SCIENCE
MASS INSTITUTE OF TECHNOLOGY
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

1

DR. BRIAN DRABBLE
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE, OR 97403

1

MR. SCOTT FOUSE
ISX CORPORATION
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

DR MICHAEL FEHLING 1
STANFORD UNIVERSITY
ENGINEERING ECO SYSTEMS
STANFORD CA 94305

RICK HAYES-ROTH 1
CIMFLEX-TEKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

MS. YOLANDA GIL 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

MR. MARK A. HOFFMAN 1
ISX CORPORATION
1165 NORTHCHASE PARKWAY
MARIETTA GA 30067

DR RON LARSEN 1
NAVAL CMD, CONTROL & OCEAN SUR CTR
RESEARCH, DEVELOP, TEST & EVAL DIV
CODE 444
SAN DIEGO CA 92152-5000

DR CRAIG KNOBLOCK 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR JOHN LOWRENCE 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. ALAN MEYROWITZ 1
NAVAL RESEARCH LABORATORY/CODE 5510
4555 OVERLOOK AVE
WASH DC 20375

ALICE MULVEHILL 1
BBN
10 MOULTON STREET
CAMBRIDGE MA 02238

DR ROBERT MACGREGOR
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292 1

DR DREW MCDERMOTT
YALE COMPUTER SCIENCE DEPT
P.O. BOX 2158, YALE STATION
51 PROPSPECT STREET
MEW HAVEN CT 06520 1

DR DOUGLAS SMITH
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304 1

DR. AUSTIN TATE
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH
80 SOUTH BRIDGE
EDINBURGH EH1 1HN - SCOTLAND 1

DIRECTOR
DARPA/ITO
3701 N. FAIRFAX DR., 7TH FL
ARLINGTON VA 22209-1714 1

DR STEPHEN F. SMITH
ROBOTICS INSTITUTE/CMU
SCHENLEY PRK
PITTSBURGH PA 15213 1

DR JONATHAN P. STILLMAN
GENERAL ELECTRIC CRD
1 RIVER RD, RM K1-5C31A
P. O. BOX 8
SCHENECTADY NY 12345 1

DR EDWARD C.T. WALKER
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138 1

DR BILL SWARTOUT
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292 1

GIO WIEDERHOLD 1
STANFORD UNIVERSITY
DEPT OF COMPUTER SCIENCE
438 MARGARET JACKS HALL
STANFORD CA 94305-2140

DR KATIA SYCARA/THE ROBOTICS INST 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIV
DOHERTY HALL RM 3325
PITTSBURGH PA 15213

DR DAVID E. WILKINS 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. PATRICK WINSTON 1
MASS INSTITUTE OF TECHNOLOGY
RM NE43-817
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

DR STEVE ROTH 1
CENTER FOR INTEGRATED MANUFACTURING
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIV
PITTSBURGH PA 15213-3890

DR YOAV SHOHAM 1
STANFORD UNIVERSITY
COMPUTER SCIENCE DEPT
STANFORD CA 94305

MR. LEE ERMAN 1
CIMFLEX TECKNOWLEDGE
1810 EMBARCARDERO RD
PALO ALTO CA 94303

DR MATTHEW L. GINSBERG 1
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403

MR JEFF GROSSMAN, CO 1
NCCOSC RDTE DIV 44
5370 SILVERGATE AVE, ROOM 1405
SAN DIEGO CA 92152-5146

DR ADELE E. HOWE
COMPUTER SCIENCE DEPT
COLORADO STATE UNIVERSITY
FORT COLLINS CO 80523 1

DR LESLIE PACK KAELBLING
COMPUTER SCIENCE DEPT
BROWN UNIVERSITY
PROVIDENCE RI 02912 1

DR SUBBARAO KAMBHAMPATI
DEPT OF COMPUTER SCIENCE
ARIZONA STATE UNIVERSITY
TEMPE AZ 85287-5406 1

DR CARLA GOMES
AFRL/IFTB
525 BROOKS RD
ROME NY 13441-4505 1

DR KAREN MYERS
AI CENTER
SRI INTERNTIONAL
333 RAVENSWOOD
MENLO PARK CA 94025 1

DR MARTHA E POLLACK
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF PITTSBURGH
PITTSBURGH PA 15260 1

DR RAJ REDDY
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213 1

DR EDWINA RISSLAND
DEPT OF COMPUTER & INFO SCIENCE
UNIVERSITY OF MASSACHUSETTS
AMHERST MA 01003 1

DR MANUELA VELOSO
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891 1

DR DAN WELD
DEPT OF COMPUTER SCIENCE & ENG
MAIL STOP FR-35
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195 1

MR JOE ROBERTS
ISX CORPORATION
4301 N FAIRFAX DRIVE, SUITE 301
ARLINGTON VA 22203 1

DR TOM GARVEY
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025 1

DIRECTOR
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

OFFICE OF THE CHIEF OF NAVAL RSCH
CODE 311
800 N. QUINCY STREET
ARLINGTON VA 22217 1

DR GEORGE FERGUSON
UNIVERSITY OF ROCHESTER
COMPUTER STUDIES BLDG, RM 732
WILSON BLVD
ROCHESTER NY 14627 1

DR STEVE HANKS
DEPT OF COMPUTER SCIENCE & ENG'G
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195 1

DR CHRISTOPHER OWENS
GTE
10 MOULTON ST
CAMBRIDGE MA 02138 1

DR JAIME CARBONNEL
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3325
PITTSBURGH PA 15213 1

DR NORMAN SADEH
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3315
PITTSBURGH PA 15213 1

DR TAIEB ZNATI
UNIVERSITY OF PITTSBURGH
DEPT OF COMPUTER SCIENCE
PITTSBURGH PA 15260 1

DR MARIE DEJARDINS
SRI INTERNATIONAL
333 RAVENSWOOD AVENUE
MENLO PARK CA 94025 1

MR.ROBERT J. KRUCHTEN
HQ AMC/SCA
203 W LOSEY ST, SUITE 1016
SCOTT AFB IL 62225-5223 1

DR. DAVE GUNNING
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

GINNY ALBERT
LOGICON ITG
2100 WASHINGTON BLVD
ARLINGTON VA 22204 1

ADAM PEASE
TECKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303 1

DR STEPHEN WESTFOLD
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304 1

DR. STEPHEN E. CROSS, DIRECTOR
SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVE 15213
PITTSBURGH PA 15213 1

DIRNSA
R509
9800 SAVAGE RD
FT MEADE MD 20755-6000
1

NSA/CSS
K1
FT MEADE MD 20755-6000
1

PHILLIPS LABORATORY
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004
1

THE MITRE CORPORATION
D460
202 BURLINGTON ROAD
BEDFORD MA 01732
1

DR. DAVID ETHERINGTON
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE, OR 97403
1

DR. MAREK RUSINKIEWICZ
MICROELECTRONCS & COMPUTER TECH
3500 WEST BALCONES CENTER DRIVE
AUSTIN, TX 78759-6509
1

MAJOR DOUGLAS DYER/ISO
DEFENSE ADVANCED PROJECT AGENCY
3701 NORTH FAIRFAX DRIVE
ARLINGTON, VA 22203-1714
1

DR. STEVE LITTLE
MAYA DESIGN GROUP
2100 WHARTON STREET S&E 702
PITTSBURGH, PA 15203-1944
1

NEAL GLASSMAN
AFOSR
110 DUNCAN AVENUE
BOLLING AFB, WASHINGTON, D.C.
29332
1

AFRL/IFT
525 BROOKS ROAD
ROME, NY 13441-4505 1

AFRL/IFTM
525 BROOKS ROAD
ROME, NY 13441-4505 1

DR. CHARLES L. MOREFIELD
ALPHATECH, INC.
2101 WILSON BLVD, SUITE 402
ARLINGTON VA 22201 1

MR. GARRY W. BARRINGER
TECHNICAL DIRECTOR
AEROSPACE CZ ISR CENTER
LANGLEY AFB VA 23665 1

DR. JAMES HENDLER
DEFENSE ADVANCED PROJECT AGENCY
3701 NORTH FAIRFAX DRIVE
ARLINGTON, VA 22203-1714 1